A Group Communication Protocol for CORBA

L. E. Moser, P. M. Melliar-Smith, R. Koch, K. Berket [*]
Department of Electrical and Computer Engineering
University of California, Santa Barbara 93106

Abstract

Group communication protocols are used in fault-tolerant systems to maintain strong replica consistency. The Fault-Tolerant Multicast Protocol (FTMP) described here is a group communication protocol specifically designed for the Common Object Request Broker Architecture (CORBA). FTMP operates over IP Multicast, and consists of the Reliable Multicast Protocol (RMP) that provides reliable source-ordered message delivery, the Reliable Ordered Multicast Protocol (ROMP) that provides reliable totally-ordered message delivery, and the Processor Group Membership Protocol (PGMP) that provides processor group membership services.

1 Introduction

Fault tolerance and high availability can be provided for the Common Object Request Broker Architecture (CORBA) [18] by means of object replication, where the replicas of an object form an object group. However, object replication is of little value unless the states of the replicas of the objects remain consistent when methods are invoked on the object group and when faults occur.

Group communication protocols [14] can facilitate the maintenance of strong replica consistency for CORBA applications by multicasting request and reply messages containing method invocations and responses, and by delivering the messages reliably in the same order to all of the members of a group. Such protocols also maintain the membership of the group.

In this paper we provide a concrete mapping of CORBA's Generalized Inter-ORB Protocol (GIOP) specification onto the Fault-Tolerant Multicast Protocol (FTMP), a multicast group communication protocol that provides reliable totally-ordered message delivery and group membership services.

[*]This research has been supported by the Defense Advanced Research Projects Agency in conjunction with the Office of Naval Research and the Air Force Research Laboratory, Rome, under Contracts N00174-95-K-0083 and F3602-97-1-0248, respectively.

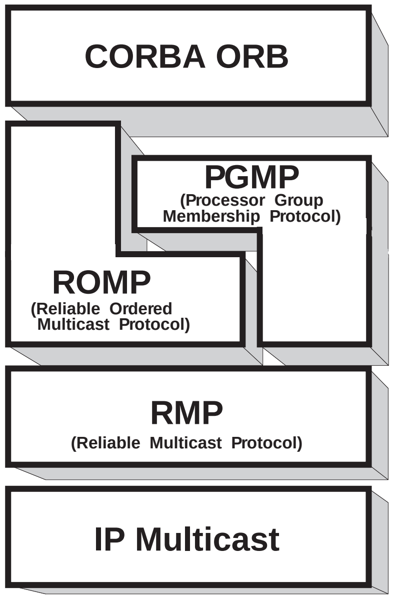
CORBA ORB
PGMP
(Processor Group Membership Protocol)
ROMP
(Reliable Ordered Multicast Protocol)
RMP
(Reliable Multicast Protocol)
IP Multicast

Figure 1: The FTMP protocol stack.

2 Overview of FTMP

The FTMP protocol stack consists of the Reliable Multicast Protocol (RMP), the Reliable Ordered Multicast Protocol (ROMP) and the Processor Group Membership Protocol (PGMP) layers, as shown in Figure 1. FTMP provides the following services:

- Messages multicast to processor groups, consisting of peer processor members, over IP Multicast
- Reliable delivery of messages to all nonfaulty members of a processor group (RMP)
- Delivery of messages in the same causal and total order to all nonfaulty members of a processor group (ROMP)
- Processor fault detection and processor group membership (PGMP).

Each FTMP message has a sequence number, a message timestamp and an acknowledgment timestamp, included in

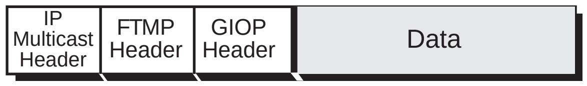

Figure 2: The encapsulation of a GIOP message.

its header. RMP uses the message sequence numbers to detect missing messages from a source. If RMP detects a missing message due to a gap in the sequence numbers, it negatively acknowledges the message. The missing message can be retransmitted by any processor that has the message.

ROMP uses the message timestamps to maintain the causal and total order of messages; the message timestamps are derived from logical Lamport clocks. Synchronized clocks can be used to achieve better performance. ROMP uses the acknowledgment timestamps for buffer management. The acknowledgment timestamp indicates that the sender has received all messages with lower timestamps from all members of the processor group to which the message is addressed.

To maintain liveness of ROMP and to keep the message delivery latency low, processors must transmit messages on a regular basis. If a processor has no application message to transmit, it transmits a Heartbeat (null) message. The Heartbeat messages also monitor the liveness of the processors and serve as a processor fault detector.

Each processor can be a member of several processor groups at the same time. PGMP adds (removes) a processor to (from) a processor group, as the fault tolerance infrastructure adds (removes) objects to (from) the object groups. The protocol removes a processor that has been convicted of being faulty from all processor groups of which it is a member.

3 FTMP Messages

3.1 Encapsulation of GIOP Messages

CORBA's Generalized Inter-ORB Protocol (GIOP) specification defines eight message types: Request, Reply, CancelRequest, LocateRequest, LocateReply, CloseConnection, MessageError and Fragment. Each such message is encapsulated in a FTMP header which, in turn, is encapsulated in an IP Multicast header, as shown in Figure 2.

3.2 FTMP Message Header

The format of the FTMP message header is:

magic	FTMP_version	byte_order	retransmission	message_type	message_size	source_processor_id

destination_processor_group_id	sequence_number	message_timestamp	ack_timestamp

- **magic** is set to FTMP
- **FTMP_version** is set to 1.0
- **byte_order** is true for little endian and false for big endian
- **retransmission** is false for the first transmission of a message and true for all subsequent retransmissions
- **message_size** is the number of bytes, including header and payload
- **message_type** is one of the types defined below
- **source_processor_id** is the identifier of the processor that originated the message
- **destination_processor_group_id** is the identifier of the processor group to which the message is multicast
- **sequence_number** is incremented each time a message that must be reliably delivered is transmitted
- **message_timestamp** is derived from the Lamport clock of the source processor and is used for message ordering
- **ack_timestamp** is a positive acknowledgment timestamp that is used for buffer management.

4 FTMP Logical Connections

Just as CORBA's Internet Inter-ORB Protocol (IIOP) maintains a physical connection between a client object and a server object with reliable source-ordered delivery of messages using TCP/IP, FTMP maintains a logical connection between a client object group and a server object group with reliable ordered delivery of messages. At most one connection is open between a client object group and a server object group at any time. Each message sent by a client (server) object group to a server (client) object group is delivered to both groups, which enables duplicate detection and suppression.

Each logical connection has an identifier that consists of the fault tolerance domain identifier and the object group identifier of the client object group, and the fault tolerance domain identifier and the object group identifier of the server object group.

The connection identifier is used in conjunction with a request number for detection of duplicate requests and duplicate replies. They are also used to match a request with its corresponding reply which is necessary, for example, when replaying messages from a log.

All of the client replicas use the same request number for a given request and all of the server replicas use the same request number for the corresponding reply. The request numbers are monotonically increasing over all connections between the two groups; therefore, each connection identifier, request number pair is unique.

Message Type	Reliable Source Ordered	Totally Ordered
Regular	Yes	Yes
RetransmitRequest	No	No
Heartbeat	No	No
ConnectRequest	No	No
Connect	Yes (except to client group)	Yes
AddProcessor	Yes (except to new member)	Yes
RemoveProcessor	Yes	Yes
Suspect	Yes	No
Membership	Yes	No

Figure 3: Message types and the delivery service provided by FTMP.

5 Reliable Message Delivery

The reliable multicast protocol (RMP) provides reliable source-ordered multicast services to the ROMP and PGMP layers of the FTMP protocol stack.

RMP employs sequence numbers to achieve reliable delivery of messages. A receiver generates negative acknowledgments for lost messages in the form of RetransmitRequests. When a processor receives a RetransmitRequest for a message that it has received, the RMP layer at that processor may retransmit the message.

As shown in Figure 3, RMP delivers Regular, Connect, AddProcessor, RemoveProcessor, Suspect and Membership messages reliably and in source order to the ROMP layer, except that it does not deliver Connect or AddProcessor messages reliably to a client group or a new member (because the client group or new member can't request retransmission to ensure reliable delivery until the connection or new membership is established). RMP delivers Heartbeat messages directly, but unreliably, to the ROMP layer as it receives them. Likewise, it delivers ConnectRequest messages directly, but unreliably, to the PGMP layer.

RMP uses three different message delivery types: Regular, RetransmitRequests and Heartbeat messages.

Regular Message. The Regular message is used to multicast GIOP messages. Regular messages are delivered reliably and in total order.

The format of the Regular message is:

FTMP_ message_header	connection_ id	request_ num	GIOP_ message

The sequence_number in the header of a Regular message, supplied by the RMP layer, contains a non-negative integer n which is the nth message in the sequence of messages multicast by the processor identified by source_processor_id to the group identified by destination_processor_group_id. RMP uses the sequence_number to detect missing messages and to maintain source-ordered delivery of messages.

The message_timestamp in the header of a Regular message, supplied by the ROMP layer, is derived from the Lamport clock of the source with the given source_processor_id.

The ack_timestamp in the header of a Regular message, supplied by the ROMP layer, indicates that the source_processor_id sending the acknowledgment has received all messages, from all processors in the processor group to which the message is addressed, that have timestamps less than or equal to the ack_timestamp.

The connection_id and request_num in the body of a Regular message are used to detect duplicate requests, duplicate replys, etc from the different replicas of an object. The request_num applies to the logical connection between a client object group and a server object group identified by the connection_id, and is different from the standard CORBA request_id which applies to a physical connection between an individual client object and an individual server object.

The connection_id and request_num are included only in the Regular messages, which are delivered to the object replicas, and not in the other FTMP message types.

RetransmitRequest Message. The RetransmitRequest message is used to request the retransmission of Regular, Connect, AddProcessor, RemoveProcessor, Suspect and Membership messages. A processor generates a RetransmitRequest message if it determines, from a gap in the sequence number of the messages that it has received from a member of a processor group, that it has not received one or more messages from that processor. No message delivery guarantees are provided for RetransmitRequest messages.

Any processor that has received a Regular, Connect, AddProcessor, RemoveProcessor, Suspect or Membership message and a corresponding RetransmitRequest message may retransmit the message. The retransmitted message is identical to the original message.

The format of the RetransmitRequest message is:

FTMP_ message_header	processor_ id	start_ seq	stop_ seq

The header of the RetransmitRequest message contains the same sequence number as the previous message originated by the sender of the RetransmitRequest message. The message_timestamp and ack_timestamp are derived from the current values provided by the ROMP layer.

The processor_id in the body of the RetransmitRequest message is the identifier of the processor for which the sender of the RetransmitRequest is missing, and requesting retransmission of, a block of messages with consecutive sequence numbers from that processor. Start_seq holds the smallest sequence number of the block of missing messages, and stop_seq holds the largest sequence number of that block. If only one message is missing, start_seq and stop_seq are equal.

Heartbeat Message. A Heartbeat message is multicast by a processor to a destination processor group if the processor has not multicast a Regular message to that group within a specified period of time. The choice of the heartbeat interval is a compromise between message latency and network traffic. A shorter heartbeat interval results in lower message latency but higher network traffic.

Heartbeat messages are delivered unreliably and in source order to the ROMP layer. The purpose of a Heartbeat message is to provide the other members of the processor group with the sender's current sequence number, message timestamp and acknowledgment timestamp. Heartbeat messages are not retransmitted, because that information would have become obsolete by the time a Heartbeat message could be retransmitted and that retransmitted message could reach a receiver.

The format of the Heartbeat message is:

FTMP_ message_header

The sequence_number in the header of the Heartbeat message is the same as the sequence number of the preceding message originated by the sender of the message. The message_timestamp and ack_timestamp are derived from the current values provided by the ROMP layer.

6 Reliable Ordered Message Delivery

The Reliable Ordered Multicast Protocol (ROMP) layer receives messages from the Reliable Multicast Protocol (RMP) layer. As shown in Figure 3, ROMP delivers Regular, Connect, AddProcessor and RemoveProcessor messages in causal and total order to the PGMP layer.

ROMP employs message timestamps, derived from logical Lamport clocks, to maintain causal and total order. A processor advances its Lamport clock so that it is always greater than the timestamp of any message that it has received or sent. Better performance can be achieved through the use of clock synchronization software, or synchronized physical clocks (*e.g.*, using Global Positioning System (GPS) satellite receivers), particularly over wide-area networks.

The ROMP layer at a processor determines when the processor no longer needs to retain a message in its buffer, because all of the processor group members have received the message and thus the processor will not need to retransmit the message. ROMP then recovers the buffer space.

7 Processor Group Membership

The Processor Group Membership Protocol (PGMP) contains mechanisms for establishing (releasing) a logical connection between two object groups. To improve efficiency, these mechanisms allow several logical connections to share the same physical connection, the same processor group and the same IP Multicast address.

PGMP also contains mechanisms for adding (removing) a processor to (from) a processor group in the case that there are no faulty processors. These mechanisms depend on the ordering of messages, which continues unaffected by the adding and removing of processors, provided that no processor is faulty. If one or more processors are faulty, the ordering of messages stops until those processors are removed from the membership. PGMP contains mechanisms to form suspicions that processors are faulty, to convict processors that enough processors suspect, and to form new memberships that exclude the convicted processors.

PGMP uses two different types of messages to establish connections between a client object group and a server object group: ConnectRequest and Connect messages. The Connect message is the server's response to the client's ConnectRequest message. Both messages are transmitted using the IP Multicast address of the fault tolerance domain of the server object group.

ConnectRequest Message. The ConnectRequest message is used by the fault tolerance infrastructure at a client to request a new connection between a client object group and a server object group. It is transmitted using the IP Multicast address of the server object group's fault tolerance domain, and is delivered to the PGMP layer. No message delivery guarantees are provided for ConnectRequest messages. The client fault tolerance infrastructure retransmits the Connect Request message periodically until the server fault tolerance infrastructure responds with a Connect message. A retransmission of the ConnectRequest message, from the client to the server, and a Connect message, from the server to the client, might cross in the network; thus, the server might receive a ConnectRequest message for a connection that it has already established. The server should ignore such requests.

The format of the ConnectRequest message is:

FTMP_ message_header	connection_ id	processor_ ids

The destination_processor_group_id, sequence_number, and message_timestamp in the header of the ConnectRequest message all have the value 0, whereas the other fields have their usual meanings and values.

The body of the ConnectRequest message contains the connection identifier and the sequence of identifiers of the processors that support the client object group.

Connect Message. The Connect message is used by the fault tolerance infrastructure at a server to establish a new connection between a client object group and a server object group. It can also be used to change the IP Multicast address or processor group used by an existing connection. When used to establish a new connection, the Connect message is transmitted using the IP Multicast address of the server object group's fault tolerance domain, to which the client is listening. When used to change the IP Multicast address or processor group for an existing connection, the Connect message is transmitted using the current IP Multicast address and the current processor group for that connection.

The Connect message is delivered reliably and in total order. However, the client processor group cannot be guaranteed reliable delivery because the members of that group are not members of the server processor group to which the Connect message is addressed. Consequently, the server processor group retransmits the Connect message periodically using the new IP Multicast address until it receives messages over the new connection.

After transmitting or receiving a Connect message, a member of the processor group is not allowed to transmit to the group any message that must be ordered, until it has received from every member of the processor group a message with a higher timestamp than the timestamp of the Connect message, such as a Heartbeat message.

If a Connect message is used to change the IP Multicast address or processor group of an existing connection, then a receiver ignores any message for the connection, that uses the current IP Multicast address and current processor group and that has a larger timestamp than that of the Connect message. The sender of such a message must retransmit the message using the new IP Multicast address and the new processor group.

The format of the Connect message is:

FTMP_message_header	connection_id	processor_group id	ip_multicast_address	timestamp_of current_membership	current_membership

The fields of the header of the Connect message are determined as for Regular messages.

The body of the Connect message contains the connection identifier, the processor group identifier, the IP Multicast address, the timestamp of the current processor group membership, and the current membership.

The timestamp_of_current_membership is the timestamp of the most recent message delivered by the processor sending the Connect message, and the current_membership is the processor group membership at that timestamp. The same interpretation is given to the timestamp_of_current_membership and current_membership for the other message types.

7.1 Group Membership Changes for Non-Faulty Processors

The AddProcessor and RemoveProcessor messages are used to add non-faulty processors to, and to remove non-faulty processors from, a processor group. When the fault tolerance infrastructure creates a new object group, or adds a member to an object group, that requires the addition of a processor to a processor group, it must ensure that any necessary change to the membership of the processor group has been completed before it makes the change to the object group membership. Correspondingly, before a processor is removed from a processor group, the fault tolerance infrastructure must remove all object replicas on that processor from their object groups. The price of this is that messages can be delivered, transiently, to processors where there are no object replicas; the benefit is a simpler processor group membership protocol.

AddProcessor Message. The AddProcessor Message is used to add a processor to a processor group. The AddProcessor message is delivered reliably and in total order.

The format of the AddProcessor message is:

FTMP_message_header	timestamp_of_current_membership	current_membership	current_sequence_numbers	new_member

The fields of the header of the AddProcessor message are determined as for a Regular message.

The body of the AddProcessor message contains the timestamp of the current membership and the current membership. It also contains the sequence number of the most recent message from each member of the current membership that has been ordered by the processor originating the message. This information allows the new member to construct the message order for messages with sequence numbers higher than those cited in the message.

RemoveProcessor Message. The RemoveProcessor message is used to remove a non-faulty processor from a processor group. The RemoveProcessor message is delivered reliably and in total order.

The format of the RemoveProcessor message is:

FTMP_message_header	member_to remove

The fields of the header of the RemoveProcessor message are determined as for a Regular message.

The body of the RemoveProcessor message contains the member to remove. The member cited in the message is removed from the membership when the RemoveProcessor message is ordered.

7.2 Group Membership Changes for Faulty Processors

The Suspect and Membership messages are used for processor group membership changes due to faulty processors. When the ROMP layer at a processor determines that another processor has crashed (because it has not received a message from that processor), it invokes PGMP to remove that processor from the processor group membership so that ROMP can order and deliver messages. The protocol then issues a fault report for the faulty processor which is conveyed to the fault tolerance infrastructure. The fault tolerance infrastructure removes the affected replicas from their object groups, and activates new or backup replicas for the object groups.

Suspect Message. The Suspect message is used to remove a faulty processor from a processor group. Suspect messages are used in conjunction with heuristic algorithms to increase the accuracy of the processor fault detectors. The Suspect message is delivered reliably and in source order, but not in total order.

The format of the Suspect message is:

FTMP_ message_header	timestamp_of_ current_membership	suspects

The fields of the header of the Suspect message are determined as for a Regular message.

The body of the Suspect message contains the timestamp of the current membership of the group and the processors that the sender of the message suspects of being faulty.

Membership Message. The Membership message is used to remove a faulty processor from a processor group. The Membership message is delivered reliably and in source order, but not in total order.

The format of the Membership message is:

FTMP_ message_header	timestamp_of_ current_membership	current_ membership	current_ sequence_numbers	new_ membership

The fields of the FTMP header of the Membership message are determined as for a Regular message.

The body of the Membership message contains the current processor group membership and the timestamp of the current membership. It also contains, for each processor in the current membership, the highest sequence number of any message addressed to the group, such that the processor sending the Membership message has received that message and all messages with smaller sequence numbers. In addition, it contains the proposed new membership.

The current sequence numbers allow a processor that survives from the current membership to the new membership to request retransmission of any message from the current membership that it has not received, but that some other processor of that membership has received. The aim is to ensure that all of the processors in the group, that survived from the current membership to the new membership, have received exactly the same messages in the current membership, which is necessary to maintain virtual synchrony.

8 Related Work

Over the past 15 years, a number of multicast group communication systems have been developed.

The Isis, Horus and Ensemble systems [3, 21] provide the services of multicast, causal multicast and atomic (total order) multicast. Those systems provide increasing flexibility in allowing the user to choose the protocol most appropriate for the application.

The Amoeba system [10] transmits messages point-to-point to a centralized sequencer, which determines the message order and then broadcasts the messages. In other sequencer-based protocols [4, 5, 9], the originators of the messages broadcast their messages.

The Trans/Total system [13] comprises the Trans protocol which provides a causal order on messages, and the Total algorithm which converts this causal order into a total order. The Transis system [1] is based on the Trans protocol and on the Isis application programmer interface. The Totem system [15] uses a logical token-passing ring to achieve robust operation and high performance.

While the above systems are oriented primarily towards local-area networks, more recent work has focused on the development of group communication systems that are scalable and oriented towards wide-area networks [19, 20].

The Atomic Group system [11] is intended for ATM networks, and is designed to support large numbers of small groups. In contrast, the InterGroup system [2] is intended for the Internet, and is designed to support large groups. The Newtop system [7] is similar in its objectives to the InterGroup system, and supports both symmetric and asymmetric ordering protocols.

Several systems have been developed that use multicast group communication protocols to augment CORBA application objects with fault tolerance and high availability.

These systems include the Electra toolkit, implemented on top of Horus, and Orbix+Isis, implemented on top of the Isis [12]. Both Electra and Orbix+Isis integrate the replication and group communication mechanisms into the ORB and require modification of the ORB.

The Eternal system [16, 17] provides fault tolerance for CORBA, using the Totem system to maintain replica consistency in a manner that is transparent to the ORB. In addition to transparency to the ORB, Eternal has the objectives of transparency to the application and ease of application programming.

The Object Group Service (OGS) [8] provides object replication through a set of services implemented on top of a ORB, including a group service, a consensus service, a monitoring service and a messaging service.

The Maestro toolkit [22] adds reliability and high availability to CORBA applications, particularly for unreplicated clients and replicated servers. The AQuA framework [6] uses the Ensemble/Maestro toolkits, as well as the Quality Objects (QuO) runtime and the Proteus dependability property manager, to provide fault tolerance for CORBA.

9 Conclusion

We have described the Fault-Tolerant Multicast Protocol (FTMP), a group communication protocol specifically designed for CORBA. The protocol consists of the Reliable Multicast Protocol (RMP) that provides reliable source-ordered multicasts, the Reliable Ordered Multicast Protocol (ROMP) that provides reliable totally-ordered multicasts, and the Processor Group Membership Protocol (PGMP) that provides processor group membership services.

References

[1] Y. Amir, D. Dolev, S. Kramer and D. Malki, ''Transis: A communication sub-system for high availability,'' *Proceedings of the IEEE 22nd Annual International Symposium on Fault-Tolerant Computing*, Boston, MA (July 1992), pp. 76-84.

[2] K. Berket, L. E. Moser and P. M. Melliar-Smith, ''The InterGroup protocols: Scalable group communication for the Internet,'' *Proceedings of IEEE GLOBECOM, Global Internet '98 Mini-Conference Record*, Sydney, Australia (November 1998), pp. 100-105.

[3] K. P. Birman and R. van Renesse, *Reliable Distributed Computing with the Isis Toolkit*, IEEE Computer Society Press, Los Alamitos, CA, 1994.

[4] J. M. Chang and N. F. Maxemchuk, ''Reliable broadcast protocols,'' *ACM Transactions on Computer Systems*, vol. 2, no. 3 (August 1984), pp. 251-273.

[5] F. Cristian and S. Mishra, ''The pinwheel asynchronous atomic broadcast protocols,'' *Proceedings of the 2nd International Symposium on Autonomous Decentralized Systems*, Phoenix, AZ (April 1995), pp. 215-221.

[6] M. Cukier, J. Ren, C. Sabnis, W. H. Sanders, D. E. Bakken, M. E. Berman, D. A. Karr and R. E. Schantz, ''AQuA: An adaptive architecture that provides dependable distributed objects,'' *Proceedings of the IEEE 17th Symposium on Reliable Distributed Systems*, West Lafayette, IN (October 1998), pp. 245-253.

[7] P. Ezhilchelvan, R. Macedo and S. Shrivastava, ''Newtop: A fault-tolerant group communication system,'' *Proceedings of the IEEE 15th International Conference on Distributed Computer Systems*, Vancouver, Canada (May 1995), pp. 296-306.

[8] P. Felber, B. Garbinato and R. Guerraoui, ''The design of a CORBA group communication service,'' *Proceedings of the IEEE 15th Symposium on Reliable Distributed Systems*, Niagra-on-the-Lake, Ontario, Canada (October 1996), pp. 150-159.

[9] W. Jia, J. Kaiser and E. Nett, ''RMP: Fault-tolerant group communication,'' *IEEE Micro*, vol. 16, no. 2 (April 1996), pp. 59-67.

[10] M. F. Kaashoek and A. S. Tanenbaum, ''Group communication in the Amoeba distributed operating system,'' *Proceedings of the IEEE 11th International Conference on Distributed Computing Systems*, Arlington, TX (May 1991), pp. 222-230.

[11] R. R. Koch, L. E. Moser and P. M. Melliar-Smith, ''The Atomic Group protocols: Group communication protocols for ATM networks,'' in preparation.

[12] S. Landis and S. Maffeis, ''Building reliable distributed systems with CORBA,'' *Theory and Practice of Object Systems*, vol. 3, no. 1 (April 1997), pp. 31-43.

[13] P. M. Melliar-Smith, L. E. Moser and V. Agrawala, ''Broadcast protocols for distributed systems,'' *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 1 (January 1990), pp. 17-25.

[14] P. M. Melliar-Smith and L. E. Moser, ''Group communication,'' *Encyclopedia of Electrical and Electronics Engineering*, vol. 8, ed. J. G. Webster, John Wiley & Sons (February 1999), pp. 500-512.

[15] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia and C. A. Lingley-Papadopoulos, ''Totem: A fault-tolerant multicast group communication system,'' *Communications of the ACM*, vol. 39, no. 4 (April 1996), pp. 54-63.

[16] L. E. Moser, P. M. Melliar-Smith and P. Narasimhan, ''Consistent object replication in the Eternal system,'' *Theory and Practice of Object Systems*, vol. 4, no. 2 (1998), pp. 81-92.

[17] P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, ''Replica consistency of CORBA objects in partitionable distributed systems,'' *Distributed Systems Engineering*, vol. 4, no. 3 (September 1997), pp. 139-150.

[18] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, Revision 2.2, OMG Technical Document formal/98-07-01 (February 1998).

[19] L. E. T. Rodrigues, H. Fonseca and P. Verissimo, ''Totally ordered multicast in large-scale systems,'' *Proceedings of the IEEE 16th International Conference on Distributed Computing Systems*, Hong Kong (May 1996), pp. 500-510.

[20] T. Tachikawa, H. Higaki, M. Takizawa, M. Gerla *et al*, ''Flexible wide-area group communication protocols -- international experiments,'' *Proceedings of the 1998 ICPP Workshop on Architectural and OS Support for Multimedia Applications*, Minneapolis, MN (August 1998), pp. 105-112.

[21] R. van Renesse, K. P. Birman, M. Hayden, A. Vaysburd *et al*, ''Building adaptive systems using Ensemble,'' *Software - Practice and Experience*, vol. 28, no. 9 (July 1998), pp. 963-979.

[22] A. Vaysburd and K. Birman, ''The Maestro approach to building reliable interoperable distributed applications with multiple execution styles,'' *Theory and Practice of Object Systems*, vol. 4, no. 2 (1998), pp. 73-80.